Addressing False Causality while Detecting Predicates in Distributed Programs

Ashis Tarafdar and Vijay K. Garg

TR-PDS-1997-011 November 1997

Parallel & Distributed Systems group
Department of Electrical & Computer Engineering
University of Texas at Austin
Austin, Texas 78712

Addressing False Causality while Detecting Predicates in Distributed Programs

Ashis Tarafdar *
ashis@cs.utexas.edu
Dept. of Computer Sciences

Vijay K. Garg †
garg@ece.utexas.edu
Dept. of Electrical and Computer Engg.

University of Texas at Austin, Austin, TX 78712

Abstract

The partial-order model of distributed computation based on the *happened before* relation has been criticized for allowing *false causality* between events in distributed programs. To address this problem, we extend this model to *strong causality diagrams* which allow independent local events, and capture the semantics of multiple local threads of control.

This paper addresses the *predicate detection problem* in the strong causality model of a distributed computation. Our focus is on the useful class of *weak conjunctive predicates*. We show that, in general, the problem is NP-complete. However, an efficient solution is demonstrated for a useful class of strong causality diagrams. Further, we show how this solution can be used to achieve an exponential reduction in the time taken to solve the general problem.

The main application of predicate detection is in debugging and testing distributed programs. Our predicate detection algorithms in the extended model can be applied to distributed debugging and testing when the processes have independent events, as in the case of multi-threaded processes.

Subject Areas: distributed software engineering and languages, distributed algorithms and verification techniques

1 Introduction

A fundamental problem in distributed systems is that of *predicate detection* [1, 6] – detecting whether a global condition occurs while running a distributed program. Its main application is in the testing, debugging and monitoring of distributed programs. Predicate detection is usually specified in a model of distributed computation based on a *happened-before* relation [9], which models the independence of concurrent events on different processes. However, it has been criticized for allowing *false causality* between events [3, 13]. This paper addresses this issue by extending the model and shows how to solve predicate detection in the extended model.

Consider running a distributed mutual-exclusion program. The happened-before model of the resulting distributed computation is shown in Figure 1(a). If mutual-exclusion violation is the predicate that we are trying to detect, it would not be detected because the message ensures that the two critical sections cannot occur at the same time. However, in actuality, the message may

*supported in part by the MCD Fellowship

†supported in part by the NSF Grants ECS-9414780, CCR-9520540, TRW faculty assistantship award, a General Motors Fellowship, and an IBM grant

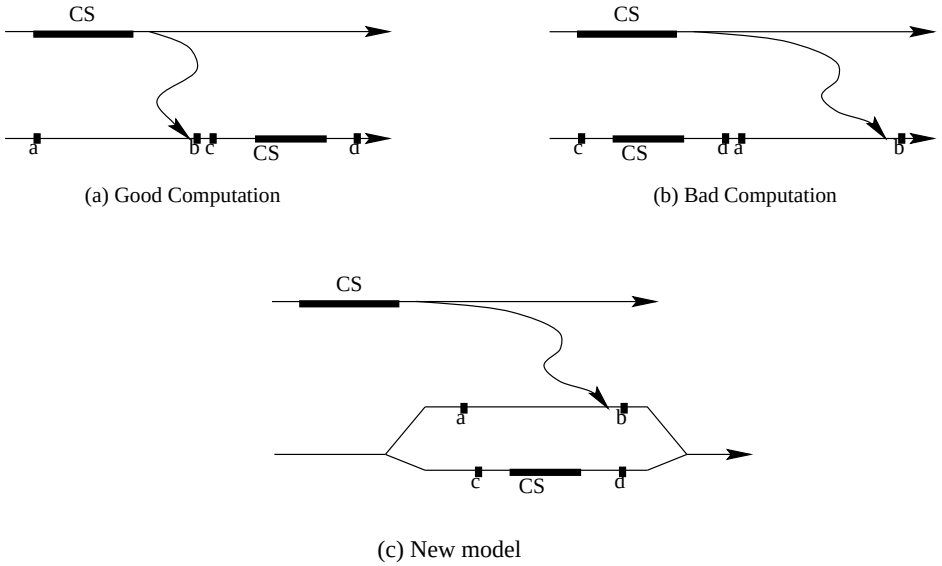

Figure 1: Example: Addressing False Causality in the happened-before model

have been fortuitous and the sections of execution marked by intervals (a, b) and (c, d) may have been independent (for example, independent threads). The scenario in Figure 1(a) may be just one possible scheduling of events. Figure 1(b) shows another scheduling in which mutual-exclusion would be violated.

A model which partially orders the events on a local process would allow events within a process to be independent. Figure 1(c) shows such a model for the example. This representation models both of the previous schedulings. In general, there would be an exponential number of happened-before representations corresponding to a single representation in the new model. We call the new model a *strong causality diagram*.

Our contributions are two-fold. First, we define and motivate a new model of computation, strong causality diagrams, that extends the happened-before model to address false causality. Next, we present results in solving predicate detection in the strong causality model. We focus on the important class of *weak conjunctive predicates* which express a large number of important global properties, and which can be solved efficiently in the happened-before model [7]. We demonstrate that for general strong causality diagrams, the problem is NP-complete. However, for certain restricted, but useful, classes of strong causality diagrams, the problem may be solved efficiently. These restricted classes correspond to having either the receive events or the send events totally ordered, while allowing all other events to be partially ordered. Further, we can decompose a general strong causality diagram into strong causality diagrams of these classes, to achieve an exponential reduction in time as compared to the naive solution. Lastly, we discuss how the strong causality diagram model may be applied in practical modeling of distributed programs with multi-threaded processes. We also discuss the applications of our results in predicate detection to testing and debugging such programs.

It is definitely harder to solve predicate detection in the strong causality model than in the happened-before model. However, even being able to efficiently solve predicate detection for a restricted class is a great improvement over the alternative of solving predicate detection on an exponential number of corresponding happened-before representations. In the worst case where the strong causality diagram does not fall into either totally ordered sends or totally ordered

receives, we may decompose the problem into strong causality diagrams each of which belongs to one of these classes and so efficiently solve for each such diagram. Even though this solution is exponential, it achieves an exponential reduction as compared to the naive approach of considering all possible happened-before representations.

In Section 2, we discuss related work. In Section 3, we formally define the strong causality diagram model and define the predicate detection problem that we will be addressing. In Section 4, we present our results in solving predicate detection in strong causality diagrams. Section 5 discusses how to apply the strong causality model and the results in predicate detection in this model. In Section 6 we summarize our conclusions.

2 Related Work

Using partial orders to model concurrency avoids the combinatorial explosion involved in the interleaving model. This fact has led to a number of studies of such partial order models [11, 9, 8, 12]. Our work builds on the happened-before model [9] by extending the same idea to allow concurrency or independence between events within a local process. We are aware of no other study of such an extended happened-before model.

Predicate detection is a widely-studied problem [1, 6]. Approaches to solving predicate detection are divided into three categories: global snapshot based [2], lattice construction based [4], and predicate restriction based [7] approaches. The first approach can detect only *stable* predicates (which remain true once they become true), and the second approach uses the interleaving model of concurrency and, therefore, suffers from the above-mentioned combinatorial explosion. We follow the last approach which uses the partial order model and limits itself to classes of predicates which can be detected efficiently. We focus on the important class of weak conjunctive predicates for which an optimal solution was provided for the happened-before model of computation [7]. Our work extends their solution to the proposed strong causality model of computation.

3 Model and Problem Definition

The usual model of a distributed computation is based on the *happened-before* relation. The motivation for this model is that a partial ordering of events in a distributed computation provides a more feasible, concise and meaningful model than a total ordering of events. We call this model a *happened-before diagram* and represent it by $(S_1, S_2, \ldots S_n, <, \rightsquigarrow)$. Here $S_1, S_2, \ldots S_n$ are sets of states on each of n processes, $<$ is the *locally precedes* relation that totally orders the states on each process in their order of occurrence in time, and $\rightsquigarrow$ is the *remotely precedes* relation that relates the state which sends a message to the state which receives it. We will use letters $s, t, u, \ldots$ to denote states.

In this model, the notion of whether a global state can "occur" is defined as follows. Define the *causally precedes* (or happened before) relation, $\rightarrow$, as the transitive closure of $<$ and $\rightsquigarrow$ (i.e. $\rightarrow = (< \cup \rightsquigarrow)^+$). If $S = S_1 \cup S_2 \cup \ldots S_n$ then $(S, \rightarrow)$ forms an irreflexive poset. If two states s and t are incomparable in this poset (i.e., $s \not\rightarrow t$ and $t \not\rightarrow s$) then s and t are called *concurrent*, denoted by $s \| t$. A *global state* is defined as a set of n states, one from each S_i. We use letters G, H to denote global states. For a global state G, $G[i]$ denotes the state in G from S_i. However, all global states cannot occur in a computation. Global states which can occur are modeled as *consistent global states* – global states in which every pair of states is concurrent.

As has been discussed, the happened-before diagrams are limited because they totally order

the events local to a process, thus, not modeling possibly independent local events. Our model, therefore, extends the locally precedes relation, $<$, from a total order to an irreflexive partial order, $<_s$, called the *strong locally precedes* relation. Our model of a distributed computation now becomes a *strong causality diagram*, $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$. We use the letters D, E, and F to represent strong causality diagrams. Note that a happened-before diagram is a special case of a strong causality diagram in which $<_s$ is a total order $<$. All notions of global states and their consistency are defined in an analogous manner for strong causality diagrams. We use the notation $\rightarrow_s$ and $\|_s$ to stand for the *strong causally precedes* and *strong concurrent* relations respectively.

A *linearization* of a partial order is a total order that contains the partial order. We define a *local linearization* of a strong causality diagram to be the happened-before diagram that may be obtained by linearizing the $<_s$ relation within each of the sets S_i, to give a $<$ relation. We represent the set of all possible local linearizations of a strong causality diagram D by $Lin(D)$. Note that, in general, a single strong causality diagram would correspond to an exponential number of happened-before diagrams. In fact, this conciseness is part of the reason for working in the extended model.

We will be interested in whether a consistent global state satisfies certain global conditions. We model these as *global predicates*, boolean functions defined on the set of all global states. Similarly, *local predicates* are boolean functions defined on the set of states in a process. We will use c to represent a global predicate and x_i to represent local predicates defined on S_i. Our focus will be on an important class of global predicates called *conjunctive predicates*. These are predicates of the form $c = x_1 \wedge x_2 \wedge \ldots x_n$ with the usual semantics (i.e., $c(G) = x_1(G[1]) \wedge x_2(G[2]) \wedge \ldots x_n(G[n])$).

We now define our general problem in this model as:

Conjunctive Predicate Detection in General Strong Causality Diagrams (CPG):
Given a strong causality diagram $D = (S_1, S_2, \ldots S_n, \rightsquigarrow, <_s)$ and a conjunctive predicate c, does there exist a consistent global state G such that $c(G)$ holds.

4 Solving Predicate Detection in Strong Causality Diagrams

The problem of weak conjunctive predicate detection was efficiently[1] and optimally solved for happened-before diagrams in [7]. However, in strong causality diagrams, the problem becomes expectedly harder. In fact, we have:

Theorem 1 *CPG is NP-complete.*

The proof[2] is by transforming 3SAT. We construct an instance of CPG with one process for each clause in 3SAT, and one state in each process for each literal in a clause. We then impose no ordering on states in a process and connect only those states representing complementary literals by $\rightsquigarrow$ edges. This guarantees that any consistent state represents a satisfying assignment for 3SAT.

The motivation for strong causality diagrams is that it concisely represents many happened-before diagrams. The following result confirms that solving predicate detection in a strong causality diagram is equivalent to solving for all of its local linearizations. First, we define the new problem:

Conjunctive Predicate Detection in Local Linearizations of General Strong Causality Diagrams (CPL):
Given a strong causality diagram $D = (S_1, S_2, \ldots S_n, \rightsquigarrow, <_s)$ and a conjunctive predicate c, does there exist a consistent global state G in any $E \in Lin(D)$ such that $c(G)$ holds.

Theorem 2 *CPG is equivalent to CPL.*

[1] in $O(mn^2)$ time, where m is a bound on the number of states in a process

[2] A complete proof may be found in the appendix.

It is easy to show[3] that any consistent global state in a local linearization of D is also consistent in D. So we are left with proving that every consistent global state G in D is a consistent global state in a local linearization of D. We construct a local linearization in which the only states that follow a $G[i]$ in the linearization of process i are ones which are strong causally preceded by some $G[j]$. It is possible to show that G is consistent in this local linearization.

This result tells us that if we exhaustively detect a predicate in each of $Lin(D)$ then we have also done so for D. Since this would be very inefficient, we identify two classes of strong dependency diagrams for which we may apply a special predicate detection algorithm to a specially chosen representative from $Lin(D)$ in order to efficiently detect a predicate.

Consider a strong causality diagram $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$. If $s \rightsquigarrow t$, then we call s a *send state* and we call t a *receive state*. Let $\mathcal{S}$ be the set of send states in S[4] and let $\mathcal{R}$ be the set of receive states in S. We use $\mathcal{S}_i$ and $\mathcal{R}_i$ to denote the set of send and receive states, respectively, in S_i. We say that D is *receive-ordered* if, for each i, the receive states in $\mathcal{R}_i$ are totally ordered by $\rightarrow_s$ (i.e., $\forall i : \forall s,t \in \mathcal{R}_i : (s \rightarrow_s t) \vee (t \rightarrow_s s)$). We say that D is *send-ordered* if, for each i, the send states in $\mathcal{S}_i$ are totally ordered by $\rightarrow_s$.

Let CPR and CPS be the CPG problem specialized to receive-ordered and send-ordered strong causality diagrams, respectively.

Let $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$ be a receive-ordered strong causality diagram. Let c be a conjunctive predicate defined on D. We now pick a special representative E from $Lin(D)$ so that it satisfies the following property:

$$\textbf{(P1)} \quad \forall i : \forall s \in S_i : \forall t \in \mathcal{R}_i : (s \parallel_s t) \Rightarrow (s \leq t)$$

This ensures that we linearize the partial order $<_s$ on each process such that a receive state is ordered after all the states that are concurrent with it. The property is well-defined because no two receive states are concurrent.

In order to ensure this property, we apply a special linearization algorithm, RECEIVE-SORT shown in Figure 2, for each process. The algorithm is a modification of a standard topological sort algorithm that gives a higher priority to non-receive states so that all states concurrent to a receive state precede it in the total ordering. The algorithm takes as input the partial order of $<_s$ restricted to a process and specified as its transitive reduction (or Hasse Diagram). It is easy to show that this correctly produces a linearization of the partial order for each process. It is also easy to show that the linearizations are those which ensure Property P1:

Theorem 3 *If $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$ is a receive-ordered strong causality diagram, and if $E = (S_1, S_2, \ldots S_n, <, \rightsquigarrow)$ is the local linearization of D produced by applying Algorithm* RECEIVE-SORT *for each process, then E satisfies Property* P1.

Proof: The following four invariants are maintained by the algorithm:
$(INV1)$ $\forall s \in S_i : k[s] =$ number of states immediately smaller than s in S_i[5] $(|\{t \in S_i : t \prec_i s\}|)$
$(INV2)$ M is the set of minimal elements in S_i that are not in $\mathcal{R}_i$
$(INV3)$ R is the set of minimal elements in S_i that are in $\mathcal{R}_i$
$(INV4)$ S_i contains no element transitively smaller than (w.r.t $\prec_i{}^+$) a minimal element

[3] The proof may be found in the appendix.
[4] S will stand for $S_1 \cup S_2 \cup \ldots S_n$ throughout this paper
[5] Here S_i is the variable used in the algorithm and not the static S_i representing all states in the ith process

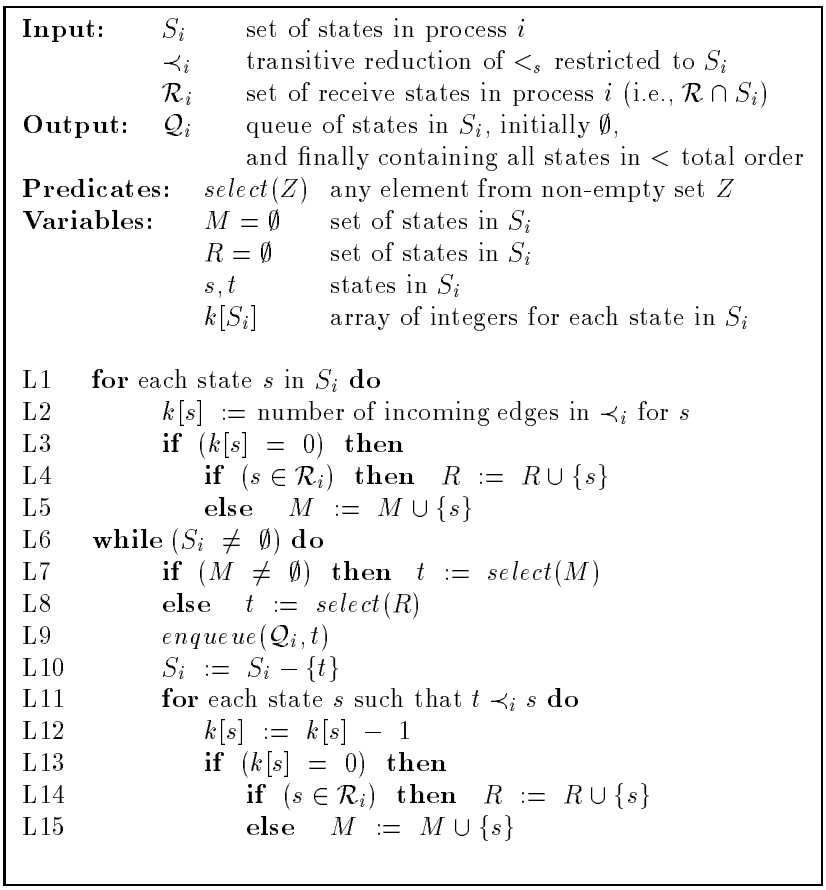

Figure 2: Algorithm RECEIVE-SORT

It is easy to prove $INV1$ from the algorithm and then prove $INV2$ and $INV3$ using it. $INV4$ follows from $INV1$ by induction on the length of the transitive chain.

Let $s \in S_i$ and $t \in \mathcal{R}_i$ be two states such that $s \parallel_s t$. Since D is receive-ordered, either $s \notin \mathcal{R}_i$ or $s = t$. If $s = t$, we are done. So we consider the case of $s \notin \mathcal{R}_i$. Focus on the iteration in which t is enqueued in $\mathcal{Q}_i$. Since $t \in \mathcal{R}_i$, t must have been chosen in line L8 (using INV2, INV3). So $M = \emptyset$ in this iteration. Further, $R = \{t\}$ (using INV3, INV4 and that D is receive-ordered). We conclude that t is the *mimimum* element of S_i in this iteration. Therefore, $s \notin S_i$ in this iteration. Therefore, it must have been enqueued before this iteration. So $s<t$. □

From Property P1, we derive the following useful property:

$$\textbf{P2} : \forall i, j : (i \neq j) : \forall s \in S_i,\ t, u \in S_j : (s \rightarrow_s t) \wedge (t \leq u) \ \Rightarrow\ (s \rightarrow_s u)$$

Lemma 1 *Let $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$ be a receive-ordered strong causality diagram and let $E \in Lin(D)$ such that $E = (S_1, S_2, \ldots S_n, <, \rightsquigarrow)$. Then:*

E satisfies Property P1 $\Rightarrow$ *E satisfies Property* P2

Proof: Let $i \neq j$ and $s \in S_i$ and $t, u \in S_j$ and let $s \rightarrow_s t$ and $(t \leq u)$. Since $s \rightarrow_s t$ and s and t are on different processes, there must be some receive state v on S_j, such that $s \rightarrow_s v$ and $v \leq_s t$.

Since $<$ is a linearization of $<_s$, we have $v \leq u$. If $v = u$, we are done. So assume $v < u$. Using the contra-positive of property P1, we have $v \not\parallel_s u$. We cannot have $u <_s v$ because $v \leq u$ and $<$ is a linearization of $<_s$. So we must have $v <_s u$. Since $s \rightarrow_s v$, we have $s \rightarrow_s u$ by transitivity. □

We now apply algorithm PRED-DETECT in Figure 3 to the special representative chosen from $Lin(D)$ using algorithm RECEIVE-SORT.

```
Input:      Q_1, Q_2, ... Q_n     queues of states in process i in < total order
            →_s                   strong causally precedes relation on states
            x_1, x_2, ... x_n     local predicates for each process
Output:     detected              boolean
Constants:  all = {1, 2, ..., n}
Variables:  low, newlow           subsets of all
            k, l                  integers in all
L1    low := all
L2    while (low ≠ ∅) do
L3        newlow := ∅
L4        for k in low do
L5            if ( ¬x_k(head(Q_k)) ) then newlow := newlow ∪ {k}
L6            else
L7                for l in all do
L7                    if (head(Q_k) →_s head(Q_l)) then newlow := newlow ∪ {k}
L8                    if (head(Q_l) →_s head(Q_k)) then newlow := newlow ∪ {l}
L9        low := newlow
L10       for k in low do
L11           deletehead(Q_k)
L12   detected := (∀k : ¬empty(Q_k))
```

Figure 3: Algorithm PRED-DETECT

Our final result shows that applying PRED-DETECT to the representative in $Lin(D)$ is sufficient for detecting the predicate in D. The main idea of the algorithm is similar to that used in [7] to optimally solve the problem for happened-before diagrams. We start with the lowest global state and move upwards. If, in a global state G, we find a state $G[i]$ that strong causally precedes another state $G[j]$ then using Property P2, we are guaranteed that $G[i]$ also strong causally precedes every state higher than $G[j]$ in the total order $<$. So $G[i]$ can be safely discarded. If no such pair of states can be found, then the global state is consistent. The algorithm discards at least one state in each iteration and so must terminate.

Theorem 4 *If $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$ is a receive-ordered strong causality diagram, c is a conjunctive predicate, and $E = (S_1, S_2, \ldots S_n, <, \rightsquigarrow)$ is a local linearization of D satisfying property* P1, *then applying algorithm PRED-DETECT to E and c solves CPR for D and c.*

Proof: Lemma 1 implies that E satisfies P2. We first prove that if PRED-DETECT returns $detected = true$ then there is indeed a strong consistent global state in D such that c holds in it. It is easy to verify the following loop invariant:

$$(INV1:)\ \forall k, l \notin low:\ \neg empty(Q_k) \wedge \neg empty(Q_l) \Rightarrow x_k(head(Q_k)) \wedge x_l(head(Q_l)) \wedge head(Q_k) \|_s head(Q_l)$$

So, if $detected = true$, then no queue is empty and the global state consisting of the heads of the queues must be strong consistent and c must hold.

We now prove that if there is strong consistent global state G such that c holds, then the algorithm must return $detected = true$. We prove by contradiction. Assume that the algorithm returns $detected = false$. Since some queue must be empty in the end, at least one state in G must be deleted in line L11. Let $G[j]$ be the state in G to be deleted first in line L11. Consider the iteration of loop L2 in which this happens. Since $c(G)$ holds, $G[j]$ could not have been added to $newlow$ in line L5. So it must have been added in line L7 or L8. In either case, $G[j] \rightarrow_s head(\mathcal{Q}_i)$, for some queue $\mathcal{Q}_i$. Since $G[j]$ is the first state in G to be deleted, $head(\mathcal{Q}_i) \leq G[i]$. So by Property P2, $G[j] \rightarrow_s G[i]$ which contradicts the strong consistency of G. □

Having established that CPR can be solved efficiently, we now address CPS. For a send-ordered strong causality diagram, we can simply make use of symmetry to transform the strong causality diagram and convert the problem to CPR. Given a send-ordered strong causality diagram $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$, we define its *inversion*, $inv(D)$ as the strong causality diagram $(S_1, S_2, \ldots S_n, \rightsquigarrow', <_s')$ where $\rightsquigarrow' = \{(s,t) |\ (t,s) \in \rightsquigarrow\}$ and $<_s' = \{(s,t) |\ (t,s) \in <_s\}$. We now have:

Theorem 5 *CPS for a send-ordered strong causality diagram $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$ and conjunctive predicate c is equivalent to CPR for $inv(D) = (S_1, S_2, \ldots S_n, \rightsquigarrow', <_s')$ and conjunctive predicate c.*

Proof: First, the set of receive states $\mathcal{R}'(S)$ for $inv(D)$ is the same as the set of send states $\mathcal{S}(S)$ for D (from the definition of $\rightsquigarrow'$). So $inv(D)$ is receive-ordered.

If G is consistent in D then for any i, j, $G[i] \not\rightarrow_s G[j]$ and $G[j] \not\rightarrow_s G[i]$. So $G[j] \not\rightarrow_s' G[i]$ and $G[i] \not\rightarrow_s' G[j]$ (from the definition of $\rightsquigarrow'$ and $<_s'$). So G is consistent in $inv(D)$.

Using a similar argument, if G is a consistent global state in $inv(D)$ then G is also consistent in D. □

If m is a bound on $|S_i|$ and e is the size of the transitive reduction of $<_s$, then we can deduce that the time complexity of applying RECEIVE-SORT to each process is $O(mn+e)$ and time complexity of PRED-DETECT is $O(mn^2)$. So the time complexity to solve CPR or CPS is $O(mn^2 + e)$.

Having efficiently solved CPR and CPS, we now take another look at the general problem CPG. We know from Theorem 1 that CPG is NP-Complete. So, it would not be prudent to search for a polynomial solution to CPG. Two naive exponential solutions are possible. Let m be a bound on $|S_i|$.The first solution enlists every global state and checks if it is consistent, a process which takes $O(m^n n^2)$ time. The second applies a predicate detection algorithm (such as in [7]) to every local linearization of the strong causality diagram, which takes $O(m^{mn} mn^2)$ time.

However, these solutions do not perform any better for strong causality diagrams which are "close" to being send-ordered or receive-ordered. For example, in a strong causality diagram which has two possible linearizations of receive states in one process, we would expect not to have to pay the full price of the above naive solutions. We now provide a solution which degrades gracefully for diagrams which are close to being send-ordered or receive-ordered.

Let k_i be a bound on the number of linearizations of the $<_s$ relation restricted to the set of receive states in S_i. If we linearize for each process, we can construct a receive-ordered strong causality diagram by adding the ordering of receive states imposed by the linearizations. For all such possible combinations of linearizations, there would be $k = k_1 \times k_2 \times \ldots k_n$ possible receive-ordered strong causality diagrams. We know from Theorem **??** that applying predicate detection to each such receive-ordered strong causality diagrams would be equivalent to applying predicate

detection to the original strong causality diagram. So we can solve CPG by applying our algorithm for CPR to k receive-ordered strong causality diagrams, taking $O(k(mn^2 + e))$. Notice that this degrades to the second naive approach in the worst case but achieves good results if k is small, or the original strong causality diagram is close to being receive-ordered. A similar approach could be used if the diagram were close to being send-ordered. Further, by decomposing into receive-ordered diagrams instead of happened-before diagrams, we save an exponential number of applications of a predicate detection algorithm as compared to the second naive approach.

5 Applications

Applying Strong Causality Diagrams:

The main application of predicate detection has been in distributed debugging and testing. A trace of the distributed computation is taken at run-time and provides the information necessary for the model of a distributed computation. Our extension to the strong causality model allows us to debug distributed programs with multi-threaded processes.

The usual practical representation of the happened-before or causally precedes relation, $\rightarrow$, has been using vector clocks [10]. Two states can then be easily checked for their $\rightarrow$ relationship by comparing their vector clocks. Since we allow partial orders on each of the processes, we must extend the vector clocks to a vector of partially ordered logical clocks [5]. Each such clock value would be an unbounded set in the general case. However, since, in practice, we can place a pre-defined bound on the number of concurrent threads, we can represent each partially ordered logical clock as a fixed-size vector. The total clock size would then be $n \times l$ where n is the number of processes and l is the maximum number of concurrent threads on a process. Given such "expanded" clocks of two states, we can check their $\rightarrow_s$ relationship in constant time (if we know which processes and threads the two states belong to).

The expanded clocks keep track of the $\rightsquigarrow$ relation by logging send and receive events. The $<_s$ relation is a little more involved because we have to decide when two states are independent. In multi-threaded processes, we can keep track of all fork and join points and inter-thread communications through shared memory as described in [5]. Although multi-threaded processes are the most direct application of strong causality diagrams, there are other types of independent events that may be identified in processes. Within the strong causality diagram model, two events are independent so long as reversing their order of execution does not change the values of any of the predicates that may be applied to them. This allows a wide variety of independences to be defined. For example, operations on independent objects are independent, and so are receives or sends on independent ports.

Applying Conjunctive Predicates:

Local predicates are any boolean functions defined on the states in a process. In practice, these states are the values of the variables defining the state of execution of the process. An example of a local predicate would be to check if a program reaches a certain function in the program text. This is equivalent to checking if the program counter variable for any thread reaches the function.

Global predicates can be any conjunctive predicates. Further, we can detect any predicates that may be reduced to conjunctive predicates. Any predicate that is a boolean expression (i.e. expression on local predicates using $\neg, \wedge, \vee$) may be converted into its disjunctive normal form and we may then apply a detection algorithm for each of the conjunctions independently. Further, any global predicate that is only satisfied by a finite set of global states may be expressed (though inefficiently) as a boolean expression of local predicates [7]. An example of a global predicate

would be detecting if functions on two processes are entered at the same time, violating a required mutual-exclusion property.

Applying Our Results:

We have provided efficient solutions to the conjunctive predicate detection problems for two classes of strong causality diagrams – receive-ordered and send-ordered. As we will now illustrate, these restrictions are met by many distributed computations in practice.

A scenario that arises very often, especially in client-server systems, is:

```
repeat
    receive a request
    create a thread to process the request and send responses
until done
```

It is clear that such a scenario is receive-ordered even though the sends and per-request processing may be independent.

Another scenario that is often used to model synchronous rounds is:

```
repeat
    receive and process messages until time = end-of-round
    send messages
until done
```

If the sends in a round occur in a fixed order or use the same port, then they are totally ordered while receives and local processing may be independent. Thus, a distributed computation resulting from such a program would be send-ordered.

Thus, very often the natural design of distributed programs involves totally ordering the sends and/or the receives. Sometimes, however, both the sends and the receives are independent, but to a small degree. Such computations would be ones where there aren't too many local linearizations of either the send states or the receive states. In such cases, we can consider all possible local linearizations of the send states or the receive states and run our algorithm on each of the resulting send-ordered or receive-ordered strong causality diagrams. Our results indicate that this would be sufficient for detecting predicates in the original computation.

6 Conclusions

We have addressed the false causality problem with the traditional happened-before model of distributed computation by extending the model to strong causality diagrams, which allow locally independent states. We have demonstrated that the conjunctive predicate detection problem is NP-complete in the new model. However, for the useful classes of send-ordered and receive-ordered classes of strong causality diagrams, efficient solutions have been designed. Further, we have shown how to decompose the general problem into problems of one of the two classes achieving an exponential reduction in time compared to the naive solution. This decomposition is useful when strong causality diagrams are "close" to being either send-ordered or receive-ordered. We have also described how to apply both our model and our results in predicate detection to the debugging and testing of distributed programs with multi-threaded processes.

References

[1] O. Babaoglu and K. Marzullo. Consistent global states of distributed systems: fundamental concepts and mechanisms. In S. Mullender, editor, *Distributed Systems*, chapter 4. Addison-Wesley, 1993.

[2] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63 – 75, February 1985.

[3] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. In *Proceedings of the 11th ACM Symposium on Operating System Principles*, pages 44 – 57. ACM, 1993.

[4] R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 163 – 173, Santa Cruz, California, 1991.

[5] C. Fidge. Logical time in distributed computing systems. *IEEE Computer*, 24(8):28 – 33, August 1991.

[6] V. K. Garg. Observation of global properties in distributed systems. In *Proceedings of the IEEE International Conference on Software and Knowledge Engineering*, pages 418 – 425, Lake Tahoe, Nevada, 1996.

[7] V. K. Garg and B. Waldecker. Detection of weak unstable predicates in distributed programs. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):299 – 307, March 1994.

[8] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem.* PhD thesis, University of Liege, 1995. also available as Vol. 1032 of Lecture Notes in Computer Science, Jan. 1996, Springer-Verlag.

[9] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558 – 565, July 1978.

[10] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms: Proc. of the International Workshop on Parallel and Distributed Algorithms*, pages 215 – 226. Elsevier Science Publishers B. V. (North Holland), 1989.

[11] A. Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to other models of concurrency, Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course, volume 255 of Lecture Notes in Computer Science*, pages 279 – 324. Springer-Verlag, 1986.

[12] V. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33 – 71, 1986.

[13] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson. Eraser: A dynamic data race detector for multi-threaded programs. In *Proceedings of the 16th ACM Symposium on Operating System Principles*. ACM, October 1997. (To be published).

A Proofs

Theorem 1 *CPG is NP-complete.*

Proof: CPG is in NP because given a global state G, $c(G)$ can be checked in polynomial time (assuming each x_i can be checked in polynomial time) and every pair of states in G can be checked for strong concurrency in polynomial time.

To show that CPG is NP-hard, we transform 3SAT to CPG. Let 3SAT be specified by $\mathcal{U}$, a set of variables, and $\mathcal{C}$, a collection of clauses. Let $k = |\mathcal{C}|$ and let $\rho_1, \rho_2, \ldots \rho_k$ be the clauses in $\mathcal{C}$. Let $\rho_i[j]$, $(1 \leq j \leq 3)$ be the 3 literals in clause ρ_i. Let $l = |\mathcal{U}|$ and let $\nu_1, \nu_2, \ldots \nu_l$ be the variables in $\mathcal{U}$. We use ν to denote any one of these variables.

Our transformation consists of defining $k + l$ processes represented by $k + l$ sets of states $S_1, S_2, \ldots S_{k+l}$. Each S_i for $1 \leq i \leq k$ corresponds to ρ_i, so that S_i contains three states $s_i[j]$, $(1 \leq j \leq 3)$, where $s_i[j]$ corresponds to $\rho_i[j]$. Each S_{k+i} for $1 \leq i \leq l$ contains two states $s_{k+i}[j]$, $(1 \leq j \leq 2)$, where $s_{k+i}[1]$ corresponds to the literal ν_i and $s_{k+1}[2]$ corresponds to the literal $\neg\nu_i$. Let $<_s = \emptyset$. Define $\rightsquigarrow$ as follows. $(s_i[k], s_j[l]) \in \rightsquigarrow, (i \neq j)$ iff $\rho_i[k] = \nu$ and $\rho_j[l] = \neg\nu$ for some variable ρ in U. Further, we define $c = x_1 \wedge x_2 \wedge \ \ldots \ x_{l+k}$ such that x_i maps all states in S_i to *true*.

First, we show that if CPG answers "yes" then 3SAT answers "yes". If CPG detects a consistent global state G, then for each variable $\nu_i \in \mathcal{U}$, exactly one of the literals ν_i and $\neg\nu_i$ can correspond to a state in G. This is because $G[k+i]$ must either correspond to ν_i or $\neg\nu_i$, using the definition of S_{k+i}, and no two states in G may correspond to ν_i and $\neg\nu_i$, using the definition of $\rightsquigarrow$. So, to every variable ν_i, we assign *true* if $G[k+i]$ corresponds to literal ν_i and *false* if it corresponds to $\neg\nu_i$. This truth assignment satisfies the literal corresponding to $G[j]$, $(1 \leq j \leq k+l)$ and, therefore, satisfies each of the clauses in $\mathcal{C}$.

Next, we show that if 3SAT answers "yes" then CPG answers "yes". Given a satisfying truth assignment to the variables in U, we construct a global state G as follows. Choose the state $G[i], (1 \leq i \leq k+l)$ so that the corresponding literal is satisfied by the truth assignment. This is possible for $G[i], (1 \leq i \leq k)$ because the truth assignment solves 3SAT and, therefore, satisfies each clause in $\mathcal{C}$. It is possible for $G[k+i], (1 \leq i \leq l)$ because either ν_i or $\neg\nu_i$ must be satisfied by the truth assignment. Every pair of states in G must be strong concurrent because of the definition of $\rightsquigarrow$ and $<_s$. G satisfies conjunctive predicate c because of the definition of c. □

Theorem 2 *CPG is equivalent to CPL.*

Proof: Let $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$ be a strong causality diagram and let c be a conjunctive predicate defined on it.

First, we prove that, if a global state G is a solution to CPL, then it is also a solution to CPG. So, let $E = (S_1, S_2, \ldots S_n, <, \rightsquigarrow)$ be a local linearization of D, and let a global state G be consistent in E, and let $c(G)$ hold. We have to prove that G is also consistent in D. Since G is consistent in E, for any two states $G[i]$ and $G[j]$, $G[i] \not\rightarrow G[j]$. Since $\rightarrow_s \subseteq \rightarrow$, we have $G[i] \not\rightarrow_s G[j]$. So G is consistent in D.

Next, we prove that, if a global state G is a solution to CPG, then it is also a solution to CPL. So, let a global state G be consistent in D, and let $c(G)$. We now construct a local linearization $E = (S_1, S_2, \ldots S_n, <, \rightsquigarrow)$ of G as follows.

For each i, define $U_i \subset S_i$ as $U_i = \{s \in S_i |\ \exists j :\ G[j] \rightarrow_s s\}$. So, U_i consists of all states in S_i for which some $G[j]$ strong causally precedes it. We now construct $<$, a local linearization of $<_s$, that orders all states in a U_i after $G[i]$ and all states not in U_i before $G[i]$. First, we define

$$<_s{}' = <_s \ \cup \ \forall i :\ \{(G[i], t) :\ t \in U_i\} \ \cup \ \{(t, G[i]) :\ (t \neq G[i]) \wedge (t \notin U_i)\}$$

We now show that $<_s'$ is antisymmetric. For the first kind of added edge, $(G[i], t)$, where $t \in U_i$, we know, from the consistency of G in D, that $t \not\rightarrow_s G[i]$. For the second kind of added edge, $(t, G[i])$, where $t \neq G[i]$ and $t \notin U_i$, we know, from the definition of U_i, that $G[i] \not\rightarrow_s t$. Therefore, $<_s'$ is antisymmetric, and so, its transitive closure is an irreflexive partial order. We define $<$ as any local linearization of the transitive closure of $<_s'$ and let $E = (S_1, S_2, \ldots S_n, <, \rightsquigarrow)$.

Lastly, we need to prove that G is consistent in E. Let U denote $U_1 \cup U_2 \cup \ldots U_n$. Consider the set V consisting of all states in S that can be reached from some $G[i]$ using edges in $<$ and $\rightsquigarrow$. We show that $V \cap G = \emptyset$. This demonstrates that G is consistent in E. If a state can be reached from $G[i]$ using $<$, then it must be in U_i (by the definition of U_i and $<$). If a state can be reached from $G[i]$ using one edge of $\rightsquigarrow$, then it must be in some U_j (by the definition of U_j). Thus, using only one edge from either $<$ or $\rightsquigarrow$ takes us to a state in U. We show that from a state in U, using $<$ and $\rightsquigarrow$, we can only reach states in U. Since $<$ orders exactly all the states in U_j higher than $G[j]$, using $<$ from a state in U keeps us within U. If we use $\rightsquigarrow$ from a state in U, then the definition of U implies that any such state would be strong causally preceded by some $G[j]$ and would belong to U. So V must be a subset of U. Since, by definition, $U \cap G = \emptyset$, we have $V \cap G = \emptyset$. This proves that G is consistent in E. $\Box$